Code de partage avec Capytale : 98b4-10266000

On utilisera les bibliothèques suivantes :

```
import numpy as np
import numpy.random as rd
import numpy.linalg as al
```

1 Exemple de constructions de graphes aléatoires

Dans ce TP, on considère des graphes d'ordre $n \in \mathbf{N}^*$ (à n sommets) simples, non orientéset on suppose que l'arête $\{i, j\}$ existe avec une probabilité p_n, pour toute paire de sommets $\{i, j\} \in \{1, ...n\}^2$, $i \neq j$
On s'intéressera surtout aux liens entre les sommets (liens sociaux dans un réseau), à savoir si on peut aller d'un sommet à l'autre, s'il y a des points isolés, ou si le graphe est connexe.
On rappelle que la matrice d'adjacence M d'un tel graphe est symétrique avec une diagonale nulle, donc construire un tel graphe revient à identifier les éléments non nuls au dessus de la diagonale de M.
On considère donc une matrice M carrée d'ordre n symétrique, de diagonale nulle, dont les éléments au dessus de la diagonale $m_{i,j}$ avec $j > i$ sont des variables aléatoires indépendantes suivant la loi de Bernoulli de paramètre p_n, et on construira le graphe de matrice d'adjacence M

1. Compléter la fonction suivante renvoyant une réalisation de M, n et p_n étant donnés en entrée :

```
def MatAlea(n, p):
    M=np.zeros((n, n))
    for i in range(...):
        for j in range(...):
            M[i, j] = ...
            M[j, i] = ...
    return M
```

2. Compléter la fonction suivante renvoyant la liste d'adjacence d'un graphe à partir de sa matrice d'adjacence M

```
def ListeAdj(M):
    n,m = np.shape(M)
    L=[]
    for i in range(...):
        voisins=[]
        for j in range(...):
            if ... :
                ...
        L.append(...)
    return L
```

3. Compléter la fonction suivante renvoyant `True` si le graphe de matrice d'adjacence M est connexe et `False` sinon.

```
def EstConnexe(M):
    n=...
    A=np.eye(n)
    for k in range(...):
        A=A+...
    for i in range(...):
        for j in range(...):
```

```
            if ... :
                return ...
    return ...
```

4. Compléter la fonction suivante renvoyant le nombre de points isolés d'un graphe (= nombre de sommets sans voisins) de liste d'adjacence L

```
def NbIsole(L):
    nombre=0
    n=...
    for i in range(...):
        if ... :
            ....
    return ...
```

5. Créer 4 graphes aléatoires d'ordre 5 avec une probabilité de connexion $p_5 = \frac{2}{5}$
Pour chacun d'eux, compter le nombre de points isolés et dire s'ils sont connexes ou non.

Pour aller plus loin : composantes connexes

6. On cherche à savoir quels sont les sommets qui sont reliés à un sommet donné x_0, c'est ce qu'on appelle une composante connexe.
L'objet de cette question est de lister toutes les composantes connexes. On remarquera que deux composantes connexes ne peuvent pas avoir de sommet commun.
Principe de l'algorithme pour obtenir une composante connexe contenant un sommet x_0 :
- initialiser trois listes, l'une `S` ne contenant qu'un seul éléments `x_0` une liste autre `C` initialement vide, et enfin une liste `Visite` de longueur `n` = ordre du graphe qui ne contient que des `False` et qui indiquera si un sommet a été visité ou non.
- à chaque étape, et jusqu'à ce que la liste `S` soit vide :
  - considérer `s` le dernier élément de `S` et le supprimer de `S`
  - si `Visite[s]==False` :
    - ▷ changer la valeur de `Visite[s]` en `True`
    - ▷ ajouter `s` élément à `C`,
    - ▷ augmenter `S` de tous les voisins `v` de `s` qui n'ont pas été visités.

Les éléments dans la liste `C` seront alors la composante connexe contenant x_0.

(a) Ecrire une fonction `Connexe(x, L)` qui renvoie la composante connexe contenant `x` d'un graphe de liste d'adjacence `L`
Tester la fonction sur les 4 graphes de la question **5** pour $x = 0$

(b) Comment tester qu'un graphe est connexe à partir de la fonction `Connexe` ?

(c) Ecrire une fonction renvoyant toutes les composantes connexes d'un graphe de liste d'adjacence `L` . Tester cette fonction sur les graphes obtenus dans la question **5**.